Una forma posible de hacer el algoritmo más eficiente es usar una técnica llamada **memoización**, que es una forma de almacenamiento en caché. La memoización consiste en guardar los resultados de las llamadas anteriores de la función en una estructura de datos, como un diccionario, y luego buscar los resultados en lugar de volver a calcularlos cuando se encuentra el mismo dato de entrada. Esto puede reducir el número de llamadas recursivas y evitar el trabajo repetido.

Para implementar la memoización para rec1(n), podemos crear un diccionario global llamado memo, e inicializarlo con el caso base de rec1(1) = 1. Luego, podemos modificar la función rec1(n) para comprobar si n ya está en memo, y devolver el valor almacenado si lo está. De lo contrario, podemos calcular el valor de rec1(n) como antes, pero también guardarlo en memo para su uso futuro. Aquí está el código modificado en Python:

```
# Diccionario global para guardar los resultados anteriores
memo = {1: 1}

# Función recursiva modificada con memoización
def rec1(n):
    # Comprobar si n ya está en memo
    if n in memo:
        # Devolver el valor almacenado
        return memo[n]
    else:
        # Calcular el valor como antes
        result = rec1(n-1) * rec1(n-1)
        # Guardar el resultado en memo
        memo[n] = result
        # Devolver el resultado
        return result
```

La complejidad temporal de este algoritmo modificado es O(n), ya que solo hace una llamada recursiva por cada valor de n desde 2 hasta n, y luego usa los valores almacenados para las llamadas posteriores. Esto es mucho mejor que O(2^n), que era la complejidad temporal del algoritmo original.